Dynamical object generation during the execution of continuous simulation models

Manuel Alfonseca, Juan de Lara, Estrella Pulido

Escuela Técnica Superior de Informática,
Universidad Autónoma de Madrid
Campus de Cantoblanco, 28049 Madrid, Spain
E-mail: {Manuel.Alfonseca, Juan.Lara, Estrella.Pulido }@ii.uam.es

Abstract. This paper describes the OOCMSP language, an object-oriented extension of CSMP, one of the most used continuous simulation languages of the seventies and eighties. The language is especially appropriate for models which can be decomposed into similar interacting components. The COOL compiler translates OOCSMP models into C++ and Java and can generate HTML skeletons which make it very easy to produce Web-based courses. With the appropriate compiler option, COOL also allows the addition and deletion of objects at execution time. The procedure is demonstrated by the implementation of a model of a geo-stationary satellite which keeps constant its distance to the Earth and a model of the inner solar system where the student can create and delete planets and study the effect of these changes on the rest of the solar system.

Keywords: Web simulation, Education, Object orientation, Java code generation, Continuous simulation

1. Introduction

System simulation [1] is one of the oldest branches of computer science. It was well advanced in the sixties, and came to maturity in the seventies. An important type of simulation is the digital continuous simulation, where time is represented by the set of multiples of a quasi-fixed time step (the elementary interval). This is equivalent to representing a continuous function by a set of samples at fixed intervals. The appropriate mathematical tool for continuous simulation is the set of algebraic-differential equations.

Continuous simulation has traditionally been programmed either in a special purpose language, or in general purpose code. Continuous simulation languages may be of different kinds, depending on their syntax:

- Block languages: each instruction represents an "electronic block", similar to

those traditionally used in analog computers [2].
- Mathematically oriented languages: the mathematical model may be used almost directly as the source program.
- Graph languages: the mathematical model is represented as a special kind of directed graph (the bond graph) [3], or by a systems dynamics graph, using the terminology and symbols proposed by J. Forrester [4].

This paper presents an object-oriented extension of CSMP, a mathematically oriented simulation language which was sponsored by IBM [5-6] and one of the most used continuous simulation languages of the seventies and eighties.

2. Object-oriented extensions in OOCSMP

Object orientation has existed since the sixties [7], but was not generally accepted until the end of the eighties, when languages as Smalltalk [8] and C++ [9] became generally used. These techniques were extended in the nineties to continuous simulation languages in two different ways:

- As libraries of classes usable from a general purpose OO language (usually C++ or Java) [10-12].
- As continuous simulation languages with built-in OO constructs [13].

We have redefined and considerably extended the CSMP language to make it object-oriented and provide it with many new facilities. Although the resulting language is a pure extension (any CSMP program may still be understood and executed by our tools), the differences are so many that a new name for the language is justified. We call it OOCSMP.

According to Peter Wegner [14], any object-oriented language should contain three main constructs and facilities:

- Objects: complex structures containing attributes (data variables) and capable of executing certain programs (functions or methods), which jointly define their behavior.
- Classes: sets of objects sharing the same attributes and behavior.
- Inheritance: classes form a hierarchy, where a class may inherit attributes and behavior from one class (simple inheritance) or several classes (multiple inheritance).

OOCSMP contains features that incorporate all three facilities:

- A class of objects may be declared with the following syntax:

```
CLASS class-name [: parent-class] {
```

```
data declaration section
[INITIAL section]
DYNAMIC [arg1 [arg2 [...]]]
  dynamic section
[method-name [arg1 [arg2 [...]]]
  method-body]
[directive section]
}
```

Simple inheritance is supported: a single parent class may be specified in the declaration of a new class.

The data declaration section contains the definition of the attributes for the objects in the class. This section is made of DATA, NAME, PARAMETER and ICON instructions, which will be described below.

The optional INITIAL section contains model equations that will be executed at the beginning of the simulation run. They are identical to the old CSMP INITIAL block, which still may be used outside a class definition.

The DYNAMIC section contains the main model equations, which should be executed once per time step. To execute these instructions, the predefined method STEP must be invoked for an object or a collection of objects. The DYNAMIC assignment instructions are similar to the same instructions outside a class definition. The DYNAMIC declaration itself, however, may contain optional parameters, which will be passed during the execution of the STEP method.

Optional additional methods (functions) applicable to objects in a class, may be defined, with or without parameters.

Typical CSMP directive instructions, such as PRINT, PLOT or FINISH, defining output requirements and abnormal end conditions for the simulation run, may be included inside a class declaration and will apply to every object in the class.

- An object in a class may be created with the following syntax:

```
class-name object-name (list-of-attribute-values)
```

  where the values of the attributes are separated by commas, and should be given in the same order in which the attributes have been declared inside the class definition. Not all the attributes need receive a value: those omitted (at the end of the list) will be assigned their default value (see below).

3. Additional extensions in OOCSMP

Besides the object-oriented extensions, many new facilities have been added to our OOCSMP language:

- New data structures: character strings, vectors, matrices and object collections.

  Character strings may be used to assign readable names to objects by means of the NAME instruction, with the following syntax:

  ```
  NAME attribute-name
  ```

  Vectors and matrices may be defined by means of the DATA instruction, which also existed in CSMP, where it was restricted to define real scalars. The extended syntax of the DATA instruction takes one of the following forms:

  ```
  DATA attribute-name
  DATA attribute-name := real-value
  DATA attribute-name[dimension]
  DATA attribute-name[dimension] := real-vector
  DATA attribute-name[index] := function(index)
  DATA attribute-name[dim1;dim2]
  DATA attribute-name[index;] := function(index)
  DATA attribute-name[;index] := function(index)
  ```

  The same vector/matrix may be declared in a form of the DATA instruction and assigned values in a different form of the instruction.

  If a value is assigned to an attribute in a DATA instruction inside a class definition, it will be considered as the default value for the attribute. The actual value may be changed when an object is declared.

  A collection of previously defined objects is declared thus:

  ```
  class-name collection-name := list-of-object-names
  ```

  where the names in the list are separated by commas.

- External data: icons and external parameters.

  Icons are external files that will be used for the graphical representation of the results of the execution of the model. Their value is a character string with the name of the associated file, and their syntax is:

  ```
  ICON attribute-name
  ```

  External parameters are data or attributes that may be introduced as program parameters at program invocation time. They should have been declared in a previous DATA instruction. If not given externally, the value specified at the

DATA instruction will be used. They are declared by means of the PARAMETER instruction, with the following syntax:

```
PARAMETER list-of-variables
```

where the names in the list are separated by commas.

- Two new assignment instructions, with the syntax:

```
variable += expression
variable -= expression
```

indicating that the value of the expression is to be added to (subtracted from) the current value of the variable. If the expression is a vector, an implicit loop is generate to add/subtract all the elements in the vector.

- Reference of attributes and functions (methods) of an object and/or a collection.

An object attribute may be referenced in the following way:

```
Object-name.attribute-name
```

These terms may be used anywhere in the program: in assignment instructions, or in the PLOT, PRINT and FINISH instructions.

The dynamic instructions of an object, all the objects in a collection, or all the objects in a class, may be executed by means of the following instructions (which may be a part of the dynamic section of a model):

```
Object-name.STEP(Argument-list)
Collection-name.STEP(Argument-list)
Class-name.STEP(Argument-list)
```

Method F, defined for a class, may be executed for an object, all the objects in a collection, or all the objects in a class, by means of the following instructions (which also may be a part of the dynamic section of a model):

```
Object-name.F(Argument-list)
Collection-name.F(Argument-list)
Class-name.F(Argument-list)
```

If F contains a term of the form Class-name.attribute-name, an implicit iteration is generated on all the objects in that class, except the one receiving the message, and an extra argument is passed to the method, that will take the value of each of these objects.

- Reuse of previously defined classes.

The INCLUDE instruction has been added to the language, with the following syntax:

```
INCLUDE "file-name"
```

Its effect is to insert the indicated file at the point of the OOCSMP program where the INCLUDE instruction appears. The included files usually contain the definition of one or more classes that the programmer wants to reuse.

- Model alternatives.

  A sentence consisting only of the symbol \ may be used to prepare different runs of the same model. All the instructions after this sentence, to the end of the program or to the next \ sentence, will be considered as a different run. The model itself does not change, but all the declarative instructions: TITLE instructions, values of parameters (DATA and TIMER instructions), object constructions, object arrays and PRINT/PLOT directives may be modified.

  Each \ sentence restores the original state of the model. All the changes specified afterwards modify that state.

- Partial differential equations.

  Models can now solve partial differential equations (which were outside the scope of CSMP). These equations may also be embodied inside a class to make it possible to generate different objects whose behavior is expressed by the same partial differential equation. The set of extensions used to declare these equations are described elsewhere [15].

- New graphical output presentations.

  To the classical CSMP PRINT and PLOT instructions, which generate listings and one-variable functional graphics, we have added the ICONICPLOT (where each variable is represented by an icon, which is repeated in proportion to the value of the variable), the CONNECTIONPLOT (used to represent equations iconically), the PLOTGRID (useful with partial differential equations), and an extended PLOT instruction that makes it possible to show surfaces and other 3D graphics. These instructions are new language constructs that integrate smoothly with the previous capabilities of CSMP.

An example

The following OOCSMP program, taken from our course at

http://www.ii.uam.es/~epulido/ecology/simul.htm is a good example of the use of the new object-oriented capabilities of OOCSMP. A class of objects describing a biological species is defined in listing 1.

```
*******************************************************
*
*           Definition of the Species class
*
*******************************************************
CLASS Species {
************************ Data ************************
 NAME name
 DATA X0, M, N1, N2:=0, start:=0, max:=1000, K:=1
 DATA Ill:=10, When:=1000, Int:=1000
************************ Equations ********************
 DYNAMIC
 X:=STEP(start)*LIMIT(1E-6,max,XT)
 XT:=INTGRL(X0,XP)
 XP:=K*X*(M-IMPULS(When,Int)*Ill)
 XPdel1 :=0
 XPdel2 :=0
 TEats  :=0
 TEats0 :=0
************************ ACTION **********************
 ACTION Species S, Percent, Last
 XPdel1 +=INSW(Percent, Percent*S.X*S.TEats0/S.TEats, 0)
 XPdel2 +=INSW(Percent, 0, Percent*S.X*S.X/S.X0)
 TEats  +=INSW(Percent, 0, 1)*S.X
 TEats0 +=INSW(Percent, 0, 1)*S.X0
 XP     +=INSW(Percent, Last*K*N2*X*XPdel1*X/X0,
                 Last*K*N1*X*XPdel2*TEats0/TEats)
}
```

Listing 1: OOCSMP definition of a species

Objects of this species may be combined to build many different ecological models, such as the one shown in listing 2.

```
INCLUDE Species.CSM
*******************************************************
*                       Actual species
*******************************************************
Species Lion   ("Lion",    2,-.02, .001        )
Species  Gnu      ("Gnu",      20,-.01,  .0001,  .017,   0,
1000, 1,  8, 100)
Species Zebra  ("Zebra",  20,-.01, .0001, .0185)
Species LGrass ("LGrass",400, .015, 0,    .00075)
Species SGrass ("SGrass",500, .015, 0,    .00075)
```

```
STEP(Lion)
STEP(Zebra)
STEP(Gnu)
STEP(LGrass)
STEP(SGrass)
ACTION(Lion,   Gnu,     0.7, 0)
ACTION(Lion,   Zebra,   0.3, 1)
ACTION(Gnu,    Lion,   -1  , 1)
ACTION(Gnu,    LGrass,  0.6, 0)
ACTION(Gnu,    SGrass,  0.4, 1)
ACTION(Zebra,  Lion,   -1  , 1)
ACTION(Zebra,  LGrass,  0.3, 0)
ACTION(Zebra,  SGrass,  0.7, 1)
ACTION(LGrass, Gnu,    -0.65, 0)
ACTION(LGrass, Zebra,  -0.35, 1)
ACTION(SGrass, Gnu,    -0.35, 0)
ACTION(SGrass, Zebra,  -0.65, 1)
*******************************************************
*                    Timer and show data
*******************************************************
TIMER delta:=0.1,FINTIM:=900,PRdelta:=10,PLdelta:=1
PLOT Lion.X, Zebra.X, Gnu.X, LGrass.X, SGrass.X, TIME
PRINT Lion.X, Zebra.X, Gnu.X, LGrass.X
METHOD ADAMS
```

Listing 2: OOCSMP model of the African savanna ecosystem

The COOL compiler

Models written in OOCSMP are compiled before execution. Our compiler, which we call COOL, alternatively generates C++ or Java code (applications or applets), and in the last case, optional html skeletons, which make it very easy to produce web-based courses with the procedure summarized by figure 1.

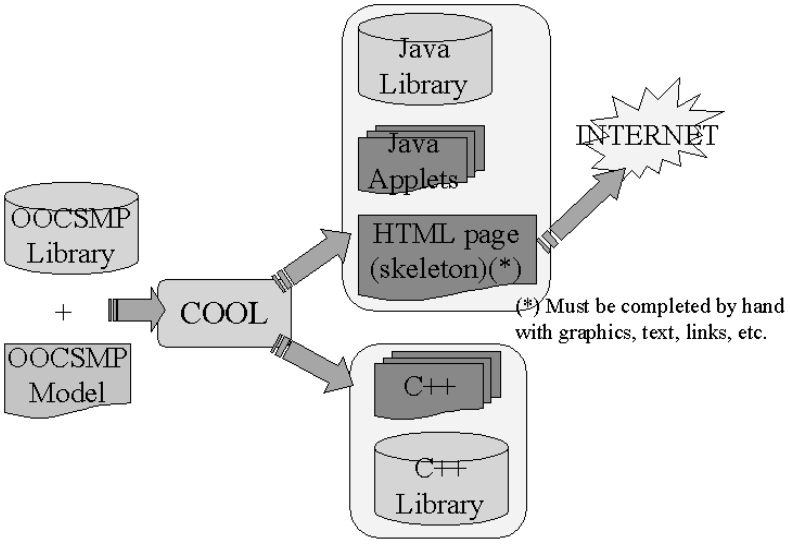


Figure 1 : Course generation procedure.

The compiler supports different options that make it possible to combine different output formats as well as model control windows (to change parameters, global data or the attributes of the objects) or timing data.

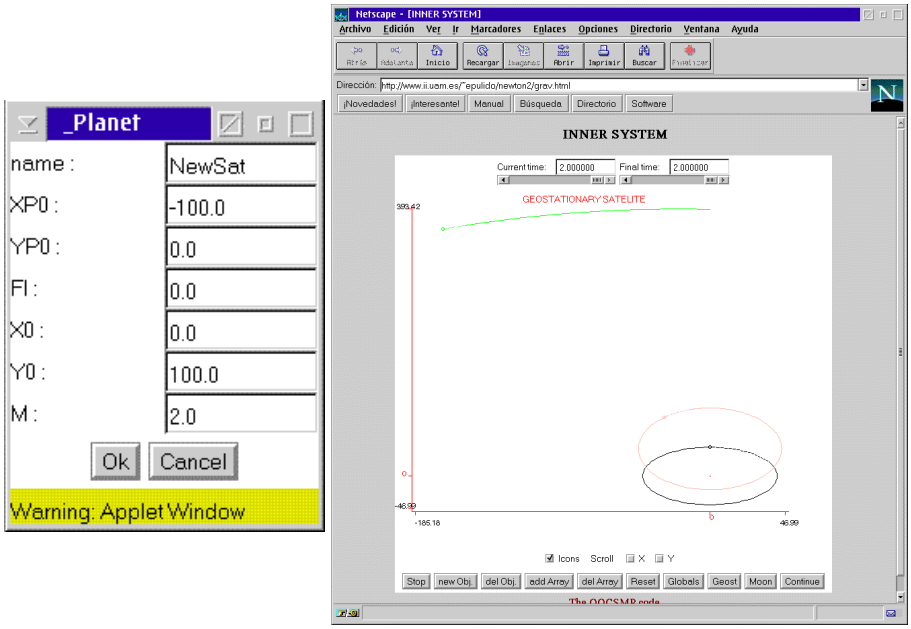
The user interface automatically generated by the compiler is very powerful, provides buttons for the different model runs (separated by the / statement) and makes it possible to add new objects to the model dynamically, as will be explained in the following section. Furthermore, it is possible to generate several presentations for the same simulation.

If the option of generating a standalone Java program is selected, a file (called `<NAME>.java`) is generated to make possible starting the application as an Applet too. If this option is not present, only the main file (called `frm_<NAME>.java`) is generated (plus all the class files). See [16] for details about the generation of educational courses with OOCSMP and COOL.

Adding objects to models during execution

The OOCSMP compiler allows the addition and deletion of objects at execution

time. As illustrated in figure 2(a), when an OOCSMP model is compiled with the appropriate compiler option the generated applet includes four additional buttons: `newObj`, `delObj`, `addArray` and `delArray`. The first two are used to create and delete objects, and the last two allow the user to add and delete objects from existing arrays.



(a) (b)

Figure 2. (a) Applet with new buttons for object creation and deletion
(b) Assigning values to object attributes

When we want to create a new object we click on the `addObj` button and we have to specify (1) the class to which the new object will belong, (2) the array to which the object will be added, and (3) the values for the object attributes. An example of the latter is shown in figure 2(b).

One example of the use of the object creation and deletion facility is the simulation of a geo-stationary satellite which keeps constant its distance to the Earth. The effect of the Moon on the satellite's orbit is illustrated by performing first a simulation of a system with two objects: one representing the Moon and the other representing the geo-stationary satellite. The second step of the experiment is to delete the Moon object and study the effect of this action on the satellite. Listing 3 shows the model

for the geo-stationary satellite in OOCSMP.

```
TITLE GEOSTATIONARY SATELLITE
DATA G:=4.979E-16, PI:=3.141592653589793
DATA MS:=5.979E21
INCLUDE "Planet.csm"
Planet Geost ("Geost",1,0,42.24637,-265.462,0,0)
Planet Moon ("Moon",7.384E19,0,392.1,-86.65,4.4,0)
Planet System := Geost, Moon
System.STEP()
System.ACTION(System)
TIMER delta:=.0005, FINTIM:=2, PRdelta:=.1,
      PLdelta:=.01
METHOD ADAMS
```

Listing 3: OOCSMP geo-stationary satellite model

A second example is a simulation of the solar system. In previous versions of the system, the student could experiment by changing the value of planet parameters and universal constants. Now the student is given the option of creating and deleting planets and studying the effect of these changes on the rest of the solar system.

The above-mentioned examples can be found at the following address:

http://www.ii.uam.es/~epulido/newton2/grav.html

Conclusion

Object orientation is very useful for the simulation of systems made of similar interconnected parts. We have used it successfully in several educational courses that use models that can be executed at a distance through the world-wide web, which may be found at the following web locations:

http://www.ii.uam.es/~epulido/newton/grav.htm
http://www.ii.uam.es/~epulido/ecology/simul.htm
http://www.ii.uam.es/~epulido/circ/modules.htm

The models have been written in OOCSMP, a language we have designed as an extension of the old CSMP. The extensions we have introduced include object-orientation and the ability to solve partial differential equations, as well as different graphical output capabilities and an advanced user interface for the model user.

Our OOCSMP compiler translates the models into C++ and/or Java, making it easy

to generate web-based courses. The compiler provides many options and makes it possible to generate extensible models: not only all parameter values may be changed at execution time, but the user may also add or delete objects, getting thus a much higher freedom to answer many new kinds of "what if" questions.
In the future, we intend to expand the possibilities offered by the dynamic modification of models at execution time, by using it in all our courses and exploring new ways of integrating it with better and more flexible user interfaces.

Acknowledgements: *This paper has been sponsored by the Spanish Interdepartmental Commission of Science and Technology (CICYT), project number TIC-96-0723-C02-01*.

References

1. Y. Monsef, "Modelling and Simulation of Complex Systems", SCS Int., Erlangen, 1997.
2. M. Alfonseca, "SIAL/71, a Continuous Simulation Compiler", in "Advances in Cybernetics and Systems", Ed. J. Rose, Gordon and Breach, London, Vol. 3, 1974, 1319-1340.
3. D. Karnopp, "Bond Graph Models for Electrochemical Energy Storage: Electrical, Chemical and Thermal Effects", Journal of the Franklin Institute, Vol 324, 1990, pp. 983-992.
4. A.A. Legasto Jr., J.W. Forrester, J.M. Lyneis, editors, "Systems Dynamics", North Holland, 1980.
5. IBM Corp.: "Continuous System Modelling Program III (CSMP III) and Graphic Feature (CSMP III Graphic Feature) General Information Manual", IBM Canada, Ontario, GH19-7000, 1972.
6. M. Alfonseca, "APL Continuous System Modelling Program: an Interactive Simulation Language", Advances in Engineering Software, Vol.1:2, 1979, 73-76.
7. O.J. Dahl, K. Nygaard, "SIMULA - An ALGOL-Based Simulation Language", Comm. ACM, 9:9, 1966, pp.671-678.
8. Digitalk Inc., "Smalltalk/V PM", Digitalk Inc., Los Angeles, 1990.
9. B. Stroustrup, "The C++ Programming Language", Addison-Wesley, Reading (Mass.), 1991-1997.
10. B. Copstein, F.R.Wagner, C.E.Pereira, "SIMOO, An Environment for the Object-Oriented Discrete Simulation", Proc. 9th European Simulation Sympossium ESS97, SCS Int., Erlangen, 1997, pp. 21-25.
11. K.J. Healy, R.A.Kilgore, "Silk: A Java-Based Process Simulation Language," Winter Simulation Conference, Atlanta, GA, 7-10 December 1997, pp. 475-482.
12. E.H. Page, R.L.Moose Jr., S.P.Griffin, “Web-Based Simulation in Simjava using Remote Method Invocation”, Winter Simulation Conference, Atlanta, GA, 7-10 December 1997,pp. 468-474.

13. H. Elmqvist, S.E.Mattson, "An Introduction to the Physical Modeling Language Modelica", Proc. 9th European Simulation Sympossium ESS97, SCS Int., Erlangen, 1997, pp. 110-114. See also http://www.Dynasim.se/Modelica/index.html.
14. P. Wegner, “Interactive foundations of object-based programming”, *Computer*, pp. 70-72, October, 1995.
15. J. de Lara, M. Alfonseca, “Simulating Partial Differential Equations in the World-Wide Web”, Proceedings of Euromedia 99, SCS Int., Munich, 1999.
16. M. Alfonseca, J. de Lara, E. Pulido, “Semiautomatic generation of educational courses in the Internet by means of an object-oriented continuous simulation language”. Proceedings of the 12th European Simulation Multiconference, Manchester (U.K.), 16-19 June 1998, pp. 547-551.